Distrib. Comput. (2014) 27:79–93
DOI 10.1007/s00446-013-0203-2

Combinatorial algorithms for distributed graph coloring

Leonid Barenboim · Michael Elkin

Received: 5 January 2012 / Accepted: 7 December 2013 / Published online: 18 December 2013

Abstract Numerous problems in Theoretical Computer Science can be solved very efficiently using powerful algebraic constructions. Computing shortest paths, constructing expanders, and proving the PCP Theorem, are just few examples of this phenomenon. The quest for combinatorial algorithms that do not use heavy algebraic machinery, but are roughly as efficient, has become a central field of study in this area. Combinatorial algorithms are often simpler than their algebraic counterparts. Moreover, in many cases, combinatorial algorithms and proofs provide additional understanding of studied problems. In this paper we initiate the study of combinatorial algorithms for Distributed Graph Coloring problems. In a distributed setting a communication network is modeled by a graph $G = (V, E)$ of maximum degree Δ. The vertices of G host the processors, and communication is performed over the edges of G. The goal of distributed vertex coloring is to color V with $(\Delta + 1)$ colors such that any two neighbors are colored with distinct colors. Currently, efficient algorithms for vertex coloring that require $O(\Delta + \log^* n)$ time are based on the algebraic algorithm of Linial (SIAM J Comput 21(1):193–201, 1992) that employs set-systems. The best currently-known combinatorial set-system free algorithm, due to Goldberg et al. (SIAM J Discret Math 1(4):434–446, 1988), requires $O(\Delta^2 + \log^* n)$ time. We significantly improve over this by devising a *combinatorial* $(\Delta + 1)$-coloring algorithm that runs in $O(\Delta + \log^* n)$ time. This exactly matches the running time of the best-known *algebraic* algorithm. In addition, we devise a tradeoff for computing $O(\Delta \cdot t)$-coloring in $O(\Delta/t + \log^* n)$ time, for almost the entire range $1 < t < \Delta$. We also compute a Maximal Independent Set in $O(\Delta + \log^* n)$ time on general graphs, and in $O(\log n / \log\log n)$ time on graphs of bounded arboricity. Prior to our work, these results could be only achieved using algebraic techniques. We believe that our algorithms are more suitable for real-life networks with limited resources, such as sensor networks.

Keywords Coloring · MIS · Set-systems

1 Introduction

1.1 Algebraic versus combinatorial algorithms

It is a common scenario in Theoretic Computer Science that very strong results are achieved by powerful non-combinatorial techniques. In many occasions consequent research focuses on devising *combinatorial* counterparts to these results. This quest for combinatorial algorithms is often justified by the desire to obtain better understanding of the problem at hand. In some cases it also leads to more efficient and simple algorithms.

One notable example of such development is the celebrated *PCP theorem*. This famous result was achieved [6,24] by algebraic techniques. Recently a significant research effort was invested in devising a combinatorial proof for this result [17,18,29,41]. Another important example is *expanders* and *Ramanujan* graphs. Near-optimal algebraic constructions of expanders were devised in [3,38,40,42,47]. Reingold,

This research is supported by the Binational Science Foundation, Grant No. 2008390. Leonid Barenboim is supported by the Adams Fellowship Program of the Israel Academy of Sciences and Humanities.

L. Barenboim (✉) · M. Elkin
Department of Computer Science, Ben-Gurion University of the Negev, POB 653, Beer-Sheva 84105, Israel
e-mail: leonidba@cs.bgu.ac.il

M. Elkin
e-mail: elkinm@cs.bgu.ac.il

Vadhan and Wigderson [49] (see also [2]) devised the first *combinatorial* construction of expanders. Though the parameters of these combinatorial constructions are somewhat inferior to algebraic ones, the techniques that were developed as a part of this effort turned out to be useful for devising a log-space S-T-connectivity algorithm [48]. Also, consequently to the work of [49], improved combinatorial constructions of expanders and near-Ramanujan graphs were devised in [13,14].

Yet another example of this phenomenon is algorithms for computing (almost) *shortest paths*. Very efficient algorithms for this problem were achieved about twenty years ago via fast matrix multiplication [5,25,50]. More recently, Aingworth et al. [1] pioneered the combinatorial approach to this problem. Consequent combinatorial algorithms for computing almost shortest paths include [12,19,20]. Notably in some scenarios, these combinatorial algorithms outperform their algebric counterparts.

This phenomenon occurs also in the area of *approximation* algorithms. Linear and semidefinite programming is an extremely powerful algebraic technique in this area. However, it is an active line of research to explore how well one can do *without linear programming*. Combinatorial approximation algorithms for telephone broadcast problem were devised in [21,22]. Oldham [43] devised combinatorial approximation algorithms for generalized flow problems. Kale and Seshadri [30] devised a combinatorial approximation algorithm for the MAX-CUT problem.

1.2 Distributed coloring

We study the *distributed coloring* problem. We are given an n-vertex unweighted undirected graph $G = (V, E)$, with each vertex $v \in V$ hosting a processor. The processors share no common memory. They communicate with each other by sending short messages (of $O(\log n)$ bits each) over the edges of E. The communication is synchronous, i.e., it occurs in discrete rounds. All vertices wake up simultaneously. Each vertex $v \in V$ has a unique identity number ($Id(v)$). For simplicity we assume that all identifiers are from the range $\{1, 2, \ldots, n\}$. All algorithms extend to larger ranges of identifiers.

Denote by Δ the maximum degree of G. In the $(\Delta + 1)$-*coloring* problem the objective is to color G with $\Delta+1$ colors *legally*, i.e., in such a way that for every edge $e = (u, v)$, the endpoints u and w will get distinct colors. The *running time* of an algorithm is the number of rounds that elapse until all vertices compute their final colors. Another closely related problem is the Maximal Independent Set (henceforth, MIS) problem. In this problem we want to compute a subset $U \subseteq V$ of independent vertices (i.e., for every $u, u' \in U$, there is no edge (u, u') in the graph) with the property that for every vertex $v \in V \setminus U$, there exists a neighbor $u \in U$ of v.

These two problems are widely considered to be among the most fundamental distributed problems. Recently, a significant progress was achieved in devising deterministic distributed algorithms for them. Specifically, the authors of the current paper [10], and independently Kuhn [32], devised a $(\Delta+1)$-coloring algorithm with running time $O(\Delta+\log^* n)$. These algorithms also directly give rise to algorithms that compute MIS within the same time. Both papers [10,32] also devised a tradeoff, and showed that for any constant ϵ, $0 < \epsilon < 1$, a $\Delta^{1+\epsilon}$-coloring can be computed in $O(\Delta^{1-\epsilon} + \log^* n)$ time. (The results in [32] are, in fact, even more general than this; they show that for any parameter t, $1 \leq t \leq \Delta$, a Δ/t-coloring can be computed in $O(\Delta \cdot t + \log^* n)$ time). Finally, on graphs of small *arboricity*,[1] the authors of the current paper devised in [9] an algorithm that computes $(\Delta+1)$-coloring and MIS in $O(\frac{\log n}{\log\log n})$ time.

All these results rely heavily on an algorithm of Linial [37], that computes an $O(\Delta^2)$-coloring within $\log^* n + O(1)$ time. The latter seminal result relies, in turn, on an algebraic construction of Erdős, Frankl and Füredi [23] of set-systems with certain special and very useful properties. Moreover, the algorithm of Kuhn [32] takes this algebraic technique one step further, and devises an algebraic construction of sets of functions that are tailored for the needs of his coloring algorithm. We remark also that even previous to the work of [10,32] $(\Delta + 1)$-coloring algorithms relied on algebraic techniques. Specifically, the $(\Delta + 1)$-coloring and MIS algorithms of Kuhn and Wattenhofer [35] that require $O(\Delta \log \Delta + \log^* n)$ time rely on Linial's algorithm. To the best of our knowledge, the best currently known deterministic algorithm of this type that does not rely on Linial's method is the algorithm due to Goldberg, Plotkin and Shannon [28]. The latter algorithm requires, however, $O(\Delta^2 + \log^* n)$ time.

The basic question that we investigate in the current paper is whether algebraic techniques are indeed necessary for devising efficient deterministic coloring and MIS algorithms. We demonstrate that it is not the case, and devise *combinatorial* (we also call them *set-system free*) coloring and MIS algorithms whose performance matches the state-of-the-art. Specifically, one of our new combinatorial algorithms (Procedure Fast-Col) computes a $(\Delta + 1)$-coloring and MIS within $O(\Delta + \log^* n)$ time. (See Theorem 4.4 and Corollary 6.1). Another one (Procedure New-Trade) provides a tradeoff and computes a $\Delta^{1+\epsilon}$-coloring in $O(\Delta^{1-\epsilon} + \log^* n)$ time, for any constant ϵ, $0 < \epsilon < 1$. (See Corollary 5.3). We also devise combinatorial $(\Delta + 1)$-coloring and MIS algorithms for graphs of small arboricity that run in $O(\frac{\log n}{\log\log n})$ time. (See Corollary 6.2. The algorithm is called Procedure New-MIS.) For a detailed list of the procedures that we devise in this paper and their running times see Table 1 below.

[1] See Sect. 2 for its definition.

Table 1 Our new set-system free procedures

Procedure	Input	Output	Running time
Generic-merge	Subgraphs $G_1, G_2, \ldots, G_\Delta$ with legal colorings $\varphi_1, \varphi_2, \ldots, \varphi_\Delta$ that employ $c_1, c_2, \ldots, c_\Delta$ colors, respectively, procedure reduce, a parameter d where $1 \leq d \leq \min\{c_i \mid 1 \leq i \leq \Delta\}$	Legal coloring of G with $(c_1 \cdot c_2 \cdot \ldots \cdot c_\Delta)/d^{\Delta-1}$ colors	$\log \Delta$ times the running time of procedure reduce
Simple-Col	The graph G	$(\Delta+1)$-coloring of G	$O(\Delta \log^2 \Delta + \log^* n)$
Mod-Trade-LEG	The graph G with an initial $\Delta^{O(1)}$-coloring ϑ	$O(\Delta^{5/4})$-coloring of G	$O(\Delta^{3/4} \log \Delta)$
Mod-Delta-LEG	The graph G with an $O(\Delta^{5/4})$-coloring of G	$(\Delta+1)$ coloring of G	$O(\Delta)$
Poly-Col	The graph G	$O(\Delta^{5/4})$-coloring of G	$O(\Delta^{3/4} \log^2 \Delta + \log^* n)$
Fast-Col	The graph G	$(\Delta+1)$-coloring of G	$O(\Delta + \log^* n)$
New-Trade	The graph G, a parameter t where $1 \leq t \leq \Delta^{1-\epsilon}$, for an arbitrarily small constant $\epsilon > 0$	$O(\Delta \cdot t)$-coloring	$O(\Delta/t + \log^* n)$
New-MIS	The graph G with $a(G) \leq \log^{1/2-\epsilon} n$, for an arbitrarily small constant $\epsilon > 0$	MIS or $(\Delta+1)$ coloring of G	$O\left(\frac{\log n}{\log \log n}\right)$

We believe that the value of these results is two-fold. First, it addresses the aforementioned question, and shows that like in the context of PCP, expanders, and computation of almost shortest paths, one also can get rid of algebraic techniques in the context of distributed graph coloring. By this our algorithms seem to uncover a new understanding of the nature of the explored problems. Second, since our algorithms are combinatorial, they are much easier for implementation by not-very-mathematically-inclined programmers. We believe that algebraic techniques may constitute a real barrier, which would scare away some of the programmers. In addition, the combinatorial nature of our algorithms enables for more efficient implementation in terms of local computation and memory requirements.[2] While the latter is usually suppressed when analyzing distributed algorithms, it may become crucial when running the algorithms on certain simple real-world communication devices, such as sensors or antennas.

1.3 Our techniques

The most tempting approach to the problem of devising combinatorial coloring algorithms is to devise a combinatorial counterpart to Linial's algorithm. However, we were not able to accomplish this. Instead we observe that some of the aforementioned coloring algorithms [9,10] start with computing an $O(\Delta^2)$-coloring via Linial's algorithm, and then employ this coloring in a way that can be relatively easily made combinatorial. Our new algorithms invoke neither the algorithm of Linial itself, nor a combinatorial analogue of it. Instead, in our algorithms we start with partitioning the edge set of the graph into Δ forests (using Panconesi-Rizzi algorithm [44]). We then color each forest separately, and combine colorings of pairs of forest. In this way we obtain an $O(\Delta^2)$-coloring of each of the $\Delta/2$ pairs of forests. Next, we employ combinatorial algorithms to manipulate the colorings in each of these pairs of forests, and to obtain, roughly speaking, a $(\Delta+1)$-coloring for each pair. We then merge these pairs again, and manipulate the resulting coloring again. We iterate in this way up until we get a unified coloring for the entire graph.

Obviously, this schematic outline suppresses many technical and somewhat involved details. Also, this scheme does not achieve our best results (Theorem 4.4), which we obtain by using more sophisticated combinatorial methods. Specifically, as a first step, we compute a $\Delta^{1+\epsilon}$-coloring, rather than $(\Delta+1)$-coloring, for some contant $\epsilon, 0 < \epsilon < 1$. We employ the scheme described above to compute this coloring in $o(\Delta + \log^* n)$ time. Then, in the second step this coloring is used as input for a recursive procedure for computing $(\Delta+1)$-coloring. This procedure (which is based on a procedure of [10], but is set-system free) partitions the graph into vertex-disjoint subgraphs with smaller degrees $d < \Delta$. Then this procedure is invoked recursively to produce $(d+1)$-colorings of these subgraphs. Finally, the resulting colorings are merged into a unified $(\Delta+1)$-coloring of the input graph.

1.4 Related work

Goldberg et al. [27] (based on [16]) devised $(\Delta+1)$-coloring and MIS algorithms that require $O(\Delta^2 + \log^* n)$

[2] Linial's [37] and Kuhn's [32] algorithms require each vertex to store locally $O(\log n + \Delta)$ words, of $O(\log n)$ bits each. Our combinatorial algorithms have lower space requirement of $O(\Delta)$ words, of $O(\log n)$ bits each.

time. Linial [37] stregthened this result, and showed that an $O(\Delta^2)$-coloring can be computed in $\log^* n + O(1)$ time. See also the work of Szegedy and Vishwanathan [51]. Kuhn and Wattenhofer [35] improved the running time of [27] to $O(\Delta \log \Delta + \log^* n)$. The latter was further improved to $O(\Delta + \log^* n)$ in [10,32]. On graphs of arboricity $a \leq \log^{1/2-\epsilon} n$, for a constant $\epsilon > 0$, [9] devised $(\Delta+1)$-coloring and MIS algorithms that require $O(\frac{\log n}{\log \log n})$ time. A variety of additional algorithms and tradeoffs for coloring graphs of small arboricity were devised in [9,11].

Awerbuch et al. [7] devised deterministic $(\Delta+1)$-coloring and MIS algorithms that require $2^{O(\sqrt{\log n \log \log n})}$ time. The latter was improved to $2^{O(\sqrt{\log n})}$ by Panconesi and Srinivasan [45]. Randomized algorithms with logarithmic running time were devised by Luby [39], and by Alon et al. [4]. Kothapalli et al. [31] showed that an $O(\Delta)$-coloring can be computed in $O(\sqrt{\log n})$ randomized time. Recently, Schneider and Wattenhofer [52] showed that $(\Delta + 1)$-coloring can be computed in randomized $O(\log \Delta + \sqrt{\log n})$ time. They have also showed a tradeoff between the number of colors and running time with very efficient algorithms for $O(\Delta+\log n)$-coloring.

The best currently-known lower bound for the $(\Delta + 1)$-coloring problem is $\Omega(\log^* n)$ due to Linial [37]. For MIS and some related problems much better lower bounds are known. Specifically, Kuhn et al. [33,34] proved that any deterministic or randomized algorithm for MIS requires $\Omega(\min\{\log \Delta, \sqrt{\log n}\})$ time.

1.5 Structure of the paper

Section 2 contains definitions and notation, and summarizes the known algorithms that are used in this paper. Section 3 presents our generic method for set-system free coloring. Section 4 contains the coloring algorithms. Section 5 contains a tradeoff between the running time and the number of colors. Section 6 presents combinatorial algorithms for additional problems.

2 Preliminaries

2.1 Definitions and notation

Unless the base value is specified, all logarithms in this paper are of base 2. The *degree* of a vertex v in a graph $G = (V, E)$, denoted *deg(v)*, is the number of edges incident to v. A vertex u such that $(u, v) \in E$ is called a *neighbor* of v in G. The maximum degree of a vertex in G, denoted $\Delta = \Delta(G)$, is defined by $\Delta(G) = \max_{v \in V} deg(v)$. A *forest* is an acyclic subgraph. A *tree* is a connected acyclic subgraph. A forest $\mathcal{F}$ can also be represented as a collection of vertex-disjoint tress $\mathcal{F} = \{T_1, T_2, \ldots, T_k\}$. A tree T is said to be oriented if (1) there is a designated *root* vertex $rt \in V(T)$, (2) every vertex $v \in V(T)$ knows whether v is the root or not, and, in the latter case, v knows which of its neighbors is the parent $\pi(v)$ of v. (The parent $\pi(v)$ of v is the unique neighbor of v that lies on the (unique) path in T connecting v with the root rt). A forest $\mathcal{F} = \{T_1, T_2, \ldots, T_k\}$ is said to be *oriented* if each of the trees $T_1, T_2, \ldots, T_k$ is oriented. A *Forest-Decomposition* of a graph $G = (V, E)$ is an edge partition such that each subgraph forms an oriented forest. The *arboricity* of a graph G is the minimal number a such that the edge set of G can be covered with at most a edge disjoint forests.

A mapping $\varphi : V \to \mathbb{N}$ is called a *coloring*. A coloring that satisfies $\varphi(v) \neq \varphi(u)$ for each edge $(u, v) \in E$ is called a *legal coloring*. For a positive integer k, a k-coloring φ is a legal coloring that employs at most k colors, i.e., for each vertex v, $\varphi(v) \in \{1, 2, \ldots, k\}$.

For two positive integers m and p, an *m-defective p-coloring* of a graph G is a (not necessarily legal) coloring of the vertices of G using p colors, such that each vertex has at most m neighbors colored by its color. Note that each color class in an m-defective coloring induces a graph of maximum degree m.

For two functions $f(n)$ and $g(n)$, we say that $f(n) = \tilde{O}(g(n))$ if $f(n) = O(g(n) \cdot \text{polylog}(g(n)))$.

2.2 Coloring procedures

In this section we summarize several well-known results that are used in the current paper. Some of our algorithms use as a black-box a procedure due to Kuhn and Wattenhofer [35]. This procedure accepts as input a graph G with maximum degree Δ, and an initial legal m-coloring, and it produces a $(\Delta + 1)$-coloring of G within time $(\Delta + 1) \cdot \lceil \log(m/(\Delta + 1)) \rceil = O(\Delta \cdot \log(m/\Delta))$. We will refer to this procedure as *KW iterative procedure*, or *KW Procedure*. For future reference we summarize this in the next lemma.

Lemma 2.1 [35] *Given a legal m-coloring of a graph with maximum degree Δ, KW procedure produces a $(\Delta + 1)$-coloring within $O(\Delta \cdot \log(m/\Delta))$ time.*

We also use a Δ-forest-decomposition procedure due to Panconesi and Rizzi [44]. (A similar construction was used also by Goldberg et al. [28]). This procedure accepts as input a graph G and computes a forest-decomposition with at most Δ forests in $O(1)$ time. We will refer to this procedure as PR *Δ-Forest-Decomposition* Procedure, or *PR Procedure*. Both these procedures are very simple. Next, we sketch them for the sake of completeness.

The KW Procedure proceeds in phases. Each phase reduces the number of employed colors by a factor of 2. This way, after $\left\lceil \log(\frac{m}{\Delta+1}) \right\rceil$ phases an m-coloring is converted into a $(\Delta + 1)$-coloring. Each phase requires $\Delta + 1$ rounds. Hence the total running time is $O(\Delta \cdot \log(\frac{m}{\Delta+1}))$. We will describe just the first phase - consequent

Table 2 Previously-known procedures [10]

Procedure	Input	Output	Running time
Delta-Col-SL	The graph G with an initial $O(\Delta^2)$-coloring	$(\Delta+1)$-coloring of G	$O(\Delta)$
Trade-Col-SL	The graph G with an initial $O(\Delta^2)$-coloring and a parameter t, $1 \leq t \leq \Delta^{1/4}$	$O(\Delta \cdot t)$-coloring of G	$O(\Delta/t)$
Defective-Col-LEG	The graph G with an initial $(c' \cdot \Delta^k)$-coloring, for some constants $c' > 0$ and $k \geq 2$, and two parameters p, q such that $0 < p^2 < q$	$\left(\frac{\log(c' \cdot \Delta^k)}{\log(q/p^2)} \cdot \Delta/p\right)$-defective p^2-coloring	$\left(\frac{\log(c' \cdot \Delta^k)}{\log(q/p^2)}\right) \cdot O(q)$

phases are very similar. The first phase starts with partitioning the palette $\{1, 2, \ldots, m\}$ into $\left\lceil \frac{m}{\Delta+1} \right\rceil$ smaller palettes $\mathcal{P}_1, \mathcal{P}_2, \ldots, \mathcal{P}_{\left\lceil \frac{m}{\Delta+1} \right\rceil}$, with $\mathcal{P}_i = \{(i-1)\cdot(\Delta+1)+1, (i-1)\cdot(\Delta+1)+2, \ldots, i\cdot(\Delta+1)\}$, for $i = 1, 2, \ldots, \left\lfloor \frac{m}{\Delta+1} \right\rfloor$, and $\mathcal{P}_{\left\lceil \frac{m}{\Delta+1} \right\rceil} = \left\{\left(\left\lceil \frac{m}{\Delta+1} \right\rceil - 1\right)\cdot(\Delta+1)+1, \ldots, m\right\}$. Next, in parallel the algorithm merges the palettes $\mathcal{P}_1$ and $\mathcal{P}_2$, $\mathcal{P}_3$ and $\mathcal{P}_4$, and so on. To merge the palettes $\mathcal{P}_1$ and $\mathcal{P}_2$ all vertices of the graph that are colored by the color $\Delta+2$ (note that $(\Delta+2) \in \mathcal{P}_2$) select for themselves in parallel an available color from $\{1, 2, \ldots, \Delta+1\}$. Then the vertices of color $\Delta+3$ do the same, and so on, all the way till color $2\cdot(\Delta+1)$. Hence after $(\Delta+1)$ rounds all vertices that were originally colored by a color from $\mathcal{P}_1 \bigcup \mathcal{P}_2$ get a color from $\{1, 2, \ldots, \Delta+1\}$; vertices that were originally colored by a color from $\mathcal{P}_3 \bigcup \mathcal{P}_4$ get a color from $\{\Delta+2, \Delta+3, \ldots, 2\cdot(\Delta+1)\}$, and so on. To summarize, after the end of the first phase the number of employed colors reduces by a factor of 2 (ignoring the integrality issues), as required.

The PR Procedure is even simpler than that. Every vertex v orients all edges (v, u) incident to v according to the identity numbers of the endpoints. Specifically, if $Id(v) < Id(u)$ then the edge $e = (v, u)$ will be oriented towards u; otherwise it will be oriented towards v. Let $\rho = \rho(v)$ be the number of edges leaving v under this orientation. Note that $\rho(v) \leq \deg(v) \leq \Delta$. The vertex v numbers all these outgoing edges by distinct numbers $1, 2, \ldots, \rho$, in an arbitrary order. It is easy to see that for each $i = 1, 2, \ldots, \Delta$, the set of edges numbered by i forms a forest, and the collection of these forests covers the entire edge set E of the graph. The only time-consuming step of this procedure is to acquire the identity numbers of neighbors; this, however, requires $O(1)$ time.

Next, we summarize the properties of several additional procedures that are used by our algorithms. In these procedures, as well as in our algorithms, we use the following naming conventions. If a procedure involves set-systems and requires a legal coloring as input, then the suffix -SL is added to its name. If the procedure involves set-systems, but does not require a coloring as input, then the suffix -SET is added to its name. If the procedure is set-system free, but it requires a legal coloring as input, then the suffix -LEG is added to its name. See Tables 1 and 2 for a concise summary of the procedures that we devise or use.

The next lemma summarizes the properties of procedures Delta-Col-SL and Trade-Col-SL for computing legal colorings, and Procedure Defective-Col-LEG for computing defective colorings, devised in [10].

Lemma 2.2 [10] (1) *Procedure Delta-Col-SL invoked on an input graph G with an initial $O(\Delta^2)$-coloring computes an $(\Delta+1)$-coloring in $O(\Delta)$ time.*

(2) *Given a graph G with an $O(\Delta^2)$ coloring, and an arbitrary parameter t, $1 < t \leq \Delta^{1/4}$, Procedure Trade-Col-SL computes an $O(\Delta \cdot t)$-coloring in time $O(\Delta/t)$.*

(3) *Procedure Defective-Col-LEG accepts as input a graph G with an initial $(c' \cdot \Delta^k)$-coloring, for some constants $c' > 0$ and $k \geq 2$, and two parameters p, q such that $0 < p^2 < q$. It computes a $(\frac{\log(c' \cdot \Delta^k)}{\log(q/p^2)} \cdot \Delta/p)$-defective p^2-coloring in time $(\frac{\log(c' \cdot \Delta^k)}{\log(q/p^2)}) \cdot O(q)$. Moreover, Procedure Defective-Col-LEG is set-system-free.*

3 The generic method

Distributed computation of a $(\Delta+1)$-coloring in general graphs is a challenging task. However, for certain graph families very efficient, and even optimal, algorithms are known. In particular, the algorithm of Goldberg and Plotkin [27] [3] is applicable for oriented forests. (Henceforth, the GP algorithm.) The GP algorithm computes a 3-coloring of a forest in $O(\log^* n)$ time. Using PR Procedure the edge set of any graph can be partitioned into Δ oriented forests. Our algorithms start by partitioning a graph into Δ forests, and computing a 3-coloring in each forest, in parallel. Then these colorings are efficiently merged into a single unified $(\Delta+1)$-coloring.

We begin with describing a procedure called *Procedure Pair-Merge* that combines the colorings of two edge-disjoint subgraphs. Procedure Pair-Merge accepts as input a graph $G = (V, E)$, such that E is partitioned into two edge-disjoint subsets E' and E''. It also accepts two legal colorings φ' and φ'' for the graphs $G' = (V, E')$ and $G'' = (V, E'')$,

[3] The algorithm of [27] is based on an earlier algorithm of Cole and Vishkin [16].

respectively. The colorings φ' and φ'' employ at most c' and c'' colors, respectively. Procedure Pair-Merge returns a new coloring φ for G such that for every $v \in V, \varphi(v) = c'' \cdot (\varphi'(v) - 1) + \varphi''(v)$. The new coloring can be seen as an ordered pair $\langle \varphi'(v), \varphi''(v) \rangle$. This completes the description of the procedure. Observe that invoking Procedure Pair-Merge requires no communication whatsoever. The properties of the procedure are summarized in the next lemma.

Lemma 3.1 *Procedure Pair-Merge produces a legal $(c' \cdot c'')$-coloring of G.*

Proof Consider an edge $e = (u, v) \in E$. The edge e belongs either to E' or to E''. If $e \in E'$ then $\varphi'(u) \neq \varphi'(v)$. Note also that $|\varphi''(u) - \varphi''(v)| \leq c'' - 1$. Consequently, $\varphi(u) = c'' \cdot (\varphi'(u) - 1) + \varphi''(u) \neq c'' \cdot (\varphi'(v) - 1) + \varphi''(v) = \varphi(v)$. If $e \in E''$ then $\varphi''(u) \neq \varphi''(v)$, and again $\varphi(u) \neq \varphi(v)$. □

Next, we describe a generic method,[4] called Generic-Merge, for merging the coloring of ℓ edge disjoint subgraphs of G, for a positive integer ℓ. Suppose that we are given an edge partition $E_1, E_2, \ldots, E_\ell$ of E. Moreover, for $1 \leq i \leq \ell$, we are given a legal coloring φ_i of $G_i = (V, E_i)$ that employs at most c_i colors, $c_i \geq \Delta + 1$. In addition, the method Generic-Merge employs an auxilary coloring procedure called Procedure Reduce(H, α, β). We will later use the method Generic-Merge with a number of different instantiations of Procedure Reduce. However, in all its instantiations Procedure Reduce accepts as input a graph H with a legal α-coloring, and computes a legal β-coloring of H. Procedure Reduce requires that $\alpha > \beta \geq \Delta + 1$. The method Generic-Merge also accepts as input a positive integer parameter d, $1 \leq d \leq \min\{c_i | 1 \leq i \leq \ell\}$. Roughly speaking, the parameter d determines the ratio between α and β. To summarize, the method Generic-Merge accepts as input the partition $E_1, E_2, \ldots, E_\ell$ of E, legal c_i-coloring φ_i of E_i, for each $i = 1, 2, \ldots, \ell$, the procedure Reduce, and the parameter d. It returns a legal coloring φ of the entire input graph G.

For future reference we summarize the required properties of a valid instantitation of Procedure Reduce.

Property 3.2 *Any instantiation of Reduce(H, α, β) must satisfy the following requirements. For any integers $\alpha > \beta \geq \Delta + 1$, and any input graph H of maximum degree Δ with an initial α-coloring, Procedure Reduce computes a legal β-coloring of H.*

The method Generic-Merge proceeds in phases. In each phase pairs of subgraphs are merged using Procedure Pair-Merge, in parallel. As a result we obtain fewer subgraphs, but a greater number of colors is employed in each subgraph. The number of colors is then reduced using Procedure Reduce, which is invoked in parallel on the merged subgraphs. This process of pairing subgraphs, merging their colorings, and reducing the number of employed colors is repeated for $\lceil \log \ell \rceil$ phases, until all subgraphs are merged into the original input graph G.

In the first phase of Generic-Merge the pairs (G_1, G_2), $(G_3, G_4), \ldots, (G_{\ell-1}, G_\ell)$ are merged into the subgraphs $G'_1, G'_2, \ldots, G'_{\lceil \ell/2 \rceil}$ by using Procedure Pair-Merge. (In other words, $G'_i = (V, E_{2i-1} \bigcup E_{2i})$, for $i = 1, 2, \ldots, \lfloor \ell/2 \rfloor$. If ℓ is odd then we define $G_{\ell+1} = (V, \emptyset)$, $c_{\ell+1} = 1$, and, consequently, $G'_{\lceil \ell/2 \rceil} = G_\ell$.) Set $\alpha_i = c'_i = c_{2i-1} \cdot c_{2i}$, and $\beta_i = \lfloor \alpha_i / d \rfloor$. (Intuitively, α_i is an upper bound on the number of colors used by the coloring that Procedure Pair-Merge produces for the subgraph G'_i. Procedure Reduce transforms this coloring into a β_i-coloring, with $\beta_i = \lfloor \alpha_i / d \rfloor$.) Next, Procedure Reduce$(G'_i, \alpha_i, \beta_i)$ is executed in parallel, for $i = 1, 2, \ldots, \lfloor \ell/2 \rfloor$. In general, a phase proceeds as follows. Suppose that the previous phase has produced the subgraphs $G'_1, G'_2, \ldots, G'_{\ell'}$, such that each subgraph is colored with at most $c'_1, c'_2, \ldots, c'_{\ell'}$ colors, respectively. Then the subgraphs G'_{2i-1} and G'_{2i} are merged into the subgraph G''_i, and Procedure Reduce$(G''_i, \alpha'_i = c'_{2i-1} \cdot c'_{2i}, \beta'_i = \lfloor c'_{2i-1} \cdot c'_{2i} / d \rfloor)$ is executed in parallel, for $i = 1, 2, \ldots, \lfloor \ell'/2 \rfloor$. If ℓ is odd then $G''_{\lceil \ell'/2 \rceil} = G'_{\ell'}$. In this case the coloring of G'_ℓ is also used for $G''_{\lceil \ell'/2 \rceil}$, instead of invoking Procedure Reduce on it. The method Generic-Merge terminates once all subgraphs are merged into a single graph, that is, the input graph G. This completes the description of the method Generic-Merge. Its pseudocode is provided below. See Figure 1 for an illustration. The properties of the method Generic-Merge are summarized in Lemma 3.3.

Algorithm 1 Method Generic-Merge($G_1, G_2, ..., G_\ell$, $\varphi_1, \varphi_2, ..., \varphi_\ell, c_1, c_2, ..., c_\ell$, Procedure Reduce, d)

```
1: /* ℓ is the number of subgraphs */
2: while ℓ > 1 do
3:   for i := 1, 2, ..., ⌊ℓ/2⌋, in parallel do
4:     {G'_i, φ'_i} := Pair-Merge(G_{2i-1}, G_{2i}, φ_{2i-1}, φ_{2i}, c_{2i-1}, c_{2i})
5:     {G_i, φ_i} := Reduce(G'_i, α_i := c_{2i-1} · c_{2i}, β_i := ⌊α_i/d⌋)
6:     c_i := ⌊c_{2i-1} · c_{2i}/d⌋
7:   end for
8:   if ℓ is odd then
9:     {G_{⌈ℓ/2⌉}, φ_{⌈ℓ/2⌉}, c_{⌈ℓ/2⌉}} := {G_ℓ, φ_ℓ, c_ℓ}
10:  end if
11:  ℓ := ⌈ℓ/2⌉
12: end while
13: return {G_1, φ_1, c_1}
```

Lemma 3.3 *The method Generic-Merge produces a legal $(c_1 \cdot c_2 \cdot \ldots \cdot c_\ell)/d^{\ell-1}$ coloring φ of the input graph G.*

Proof First, we prove that the coloring φ is legal. We prove by induction on the number of phases that each subgraph is legally colored in each phase. For the base case recall that we assumed that all subgraphs of the initial parti-

[4] We refer to a procedure as *generic method* if it accepts another procedure as input.

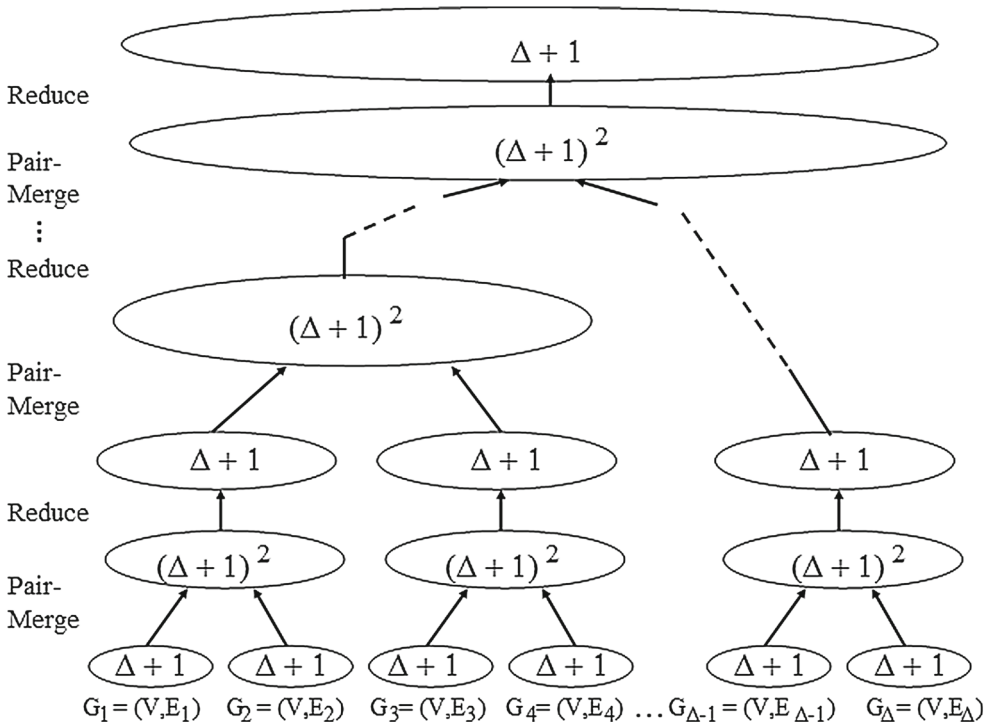

Fig. 1 Execution of the method Generic-Merge on the edge partition $\{G_1, G_2, \ldots, G_\Delta\}$ of G. The expressions inside the ovals represent the number of colors used. Initially, each subgraph $G_1, G_2, \ldots, G_\Delta$ is colored with $\Delta + 1$ colors. In this example the employed Procedure Reduce accepts as input a $(\Delta + 1)^2$-coloring and transforms it into a $(\Delta + 1)$-coloring

tion $G_1, G_2, \ldots, G_\ell$ are legally colored by the colorings $\varphi_1, \varphi_2, \ldots, \varphi_\ell$, respectively. For the induction step assume that at the beginning of a phase i the partition of the input graph consists of the subgraphs $G'_1, G'_2, \ldots, G'_{\ell'}$. In phase i the subgraphs are merged using procedure Pair-Merge that produces legal coloring in each merged subgraph. (See Lemma 3.1.) Next, Procedure Reduce is invoked on the resulting subgraphs $G''_1, G''_2, \ldots, G''_{\lfloor \ell'/2 \rfloor}$, producing a legal coloring in each of them. Consequently, at the beginning of the next phase $i + 1$ the subgraphs $G''_1, G''_2, \ldots, G''_{\lceil \ell'/2 \rceil}$ are legally colored. Hence after $\lceil \log \ell \rceil$ phases all subgraphs are merged into the input graph G, and at this point G is legally colored.

Next, we prove that φ employs at most $(c_1 \cdot c_2 \cdot \ldots \cdot c_\ell)/d^{\ell-1}$ colors. In order to merge ℓ subgraphs into a single one, exactly $\ell - 1$ executions of Pair-Merge are performed during the invocation of the method Generic-Merge. (Because each execution of Procedure Pair-Merge reduces the number of subgraphs in the partition by 1.) An execution of Procedure Reduce follows each execution of Procedure Pair-Merge. Hence there are also exactly $\ell - 1$ executions of Procedure Reduce. Initially, the product of the number of colors used in all subgraphs of the partition is $c_1 \cdot c_2 \cdot \ldots \cdot c_\ell$. After each invocation of Procedure Reduce this product is reduced by a multiplicative factor of at least d. Once Generic-Merge terminates all subgraphs are merged into a single graph that is colored with at most $(c_1 \cdot c_2 \cdot \ldots \cdot c_\ell)/d^{\ell-1}$ colors. □

4 Coloring algorithms

In this section we present several procedures for computing a $(\Delta + 1)$-coloring. We begin with the simplest procedure, called *Procedure Simple-Col*. This procedure, however, requires an $O(\Delta \log^2 \Delta + \log^* n)$ time. Then in Sect. 4.2 we devise a much more efficient procedure, called *Procedure Poly-Col*. This procedure requires $o(\Delta) + O(\log^* n)$ time, but employs more than $\Delta + 1$ colors. Finally, in Sect. 4.3 we devise a procedure, called *Proceure Fast-Col*, that uses the other procedures and computes a $(\Delta + 1)$-coloring within $O(\Delta + \log^* n)$ time.

4.1 Procedure Simple-Col

In this section we present our first algorithm that employs the method Generic-Merge, called *Procedure Simple-Col*. The method Generic-Merge accepts as input a partition of the input graph such that each subgraph in the partition is legally colored. In order to compute such a partition, we invoke the PR Procedure. Recall that this procedure computes a Δ-forest-decomposition of G. In other words, the procedure outputs an edge partition $\{G_1, G_2, \ldots, G_\Delta\}$, $G_i = (V, E_i)$, $i \in \{1, 2, \ldots, \Delta\}$, such that each subgraph in this partition is an oriented forest. Next, each forest is colored with 3 colors using the GP algorithm. The Δ invocations of the GP algorithm are performed in parallel. They result in

legal colorings $\varphi_1, \varphi_2, \ldots, \varphi_\Delta$. Since $\Delta \geq 2$, each coloring φ_i, $i = 1, 2, \ldots, \Delta$, employs at most $\Delta + 1$ colors.

Recall that the method Generic-Merge also accepts as input parameter a procedure (Procedure Reduce) for reducing the number of colors in a given coloring of a graph. In Procedure Simple-Col we employ the KW iterative procedure as Procedure Reduce. (The KW iterative procedure accepts as input a graph H with an α-coloring, $\alpha \geq \Delta + 1$, and computes a $(\Delta + 1)$-coloring of H in time $O(\Delta \log(\alpha/\Delta))$. See Sect. 2). We invoke Generic-Merge on the partition $\{G_1, G_2, \ldots, G_\Delta\}$ with the colorings $\varphi_1, \varphi_2, \ldots, \varphi_\Delta$ such that $c_1 = c_2 = \cdots = c_\Delta = \Delta + 1$. Finally, the parameter d is set to $\Delta + 1$. Consequently, in each phase of Generic-Merge pairs of $(\Delta + 1)$-colored subgraphs are merged into $(\Delta + 1)^2$-colored subgraphs, and then the number of colors in each subgraph is reduced back to $\Delta + 1$. Once Generic-Merge terminates, the input graph is legally colored using $\Delta + 1$ colors. The pseudocode of Procedure Simple-Col is given below. Its properties are summarized in Theorem 4.1.

Algorithm 2 Procedure Simple-Col(G)

1: $\{G_1, G_2, ..., G_\Delta\}$:= Δ-Forests-Decomposition(G) /*using PR Procedure */
2: **for** $i = 1, 2, ..., \Delta$ in parallel **do**
3: φ_i := 3-color(G_i) /*using GP algorithm */
4: $c_i := \Delta + 1$
5: **end for**
6: $d := \Delta + 1$
7: Generic-Merge($G_1, G_2, ..., G_\Delta, \varphi_1, \varphi_2, ..., \varphi_\Delta, c_1, c_2, ..., c_\Delta$, KW-iterative procedure, d)

Theorem 4.1 *Procedure Simple-Col produces a legal $(\Delta + 1)$-coloring of G. Its running time is $O(\Delta \log^2 \Delta) + \log^* n$. This procedure is set-system free.*

Proof First, we prove that procedure Simple-Col produces a legal $(\Delta+1)$-coloring of G. The partition P that is passed to Generic-Merge satisfies that each subgraph G_i in the partition is legally colored by φ_i with at most $c_i = \Delta+1$ colors. Each time the KW iterative procedure is executed, it accepts as input a subgraph H of G that is colored with at most $(\Delta+1)^2$ colors. (The subgraph H is a result of merging two subgraphs colored with at most $\Delta + 1$ colors each.) An invocation of the KW iterative procedure on H produces a legal $(\Delta + 1)$-coloring of H. Hence, the KW iterative procedure is a correct instantiation for Procedure Reduce. (See Property 3.2.) By Lemma 3.3, the method Generic-Merge produces a legal coloring with at most $(\Delta + 1)^\ell/(\Delta + 1)^{\ell-1} = (\Delta + 1)$ colors.

Next, we prove that the running time of Procedure Simple-Col is $O(\Delta \log^2 \Delta)+\log^* n$. The running time of the PR Procedure for computing the forest-decomposition in step 1 is $O(1)$. The time required for computing 3-colorings by the GP algorithm is $\log^* n+O(1)$. Next, we analyze the running time of the method Generic-Merge when it is executed in conjunction with the KW iterative procedure. Each execution of the KW iterative procedure on a subgraph H reduces the number of colors used for coloring H from $(\Delta+1)^2$ to $\Delta+1$. Hence, by Lemma 2.1, the running time of each invocation of the KW iterative procedure is $O(\Delta \log \Delta)$. In each phase all invocations are performed in parallel. Hence, each phase requires $O(\Delta \log \Delta)$ time. There are $\lceil \log \Delta \rceil$ phases. Therefore, the running time of Generic-Merge is $O(\Delta \log^2 \Delta)$. Hence, the overall running time is $O(\Delta \log^2 \Delta) + \log^* n$. □

4.2 Procedure Poly-Col

Our algorithm consists of two major stages. In the first stage we compute a $\Delta^{O(1)}$-coloring, and in the second stage we reduce the number of colors from $\Delta^{O(1)}$ to $(\Delta + 1)$. In the existing $(\Delta + 1)$-coloring algorithms that run in $O(\Delta) + \log^* n$ time [10,32], both these stages employ set systems. In fact, as far as we know, currently there is no known set-system free $\Delta^{O(1)}$-coloring algorithm that runs within $O(\Delta) + \log^* n$ time. The situation is somewhat better in the context of the second stage, as there is a known set-system free algorithm (KW Procedure) that accepts a $\Delta^{O(1)}$-coloring as input and returns a $(\Delta+1)$-coloring. However, its running time ($O(\Delta \log \Delta)$) is higher than the desired bound of $O(\Delta) + \log^* n$. Therefore, we speed up both the first and the second stages of the aforementioned scheme, and achieve a set-system free $(\Delta + 1)$-coloring algorithm with running time $O(\Delta + \log^* n)$.

Before we begin with the description of our new algorithm, we provide a brief survey of several known (not set-system free) algorithms (due to [10]) that employ the two-stage technique described above. In the sequel, we modify these algorithms, and eliminate the steps that employ set-systmes. Then we employ these modified versions for devising our new results.

We start with sketching Procedure Defective-Col-SET from [10], that accepts as input a graph G and two parameters p and q, and returns a defective coloring of G. This procedure starts (line 1 of Algorithm 3) with computing an $O(\Delta^2)$-coloring ϑ using an algebraic algorithm of Linial [37]. Then (line 2 of Algorithm 3) it employs a subroutine, Procedure Defective-Col-LEG, that converts ϑ into an $O(\frac{\log(\Delta^2)}{\log(q/p^2)} \cdot \Delta/p)$-defective p^2-coloring ψ of G.

Algorithm 3 Procedure Defective-Col-SET (G, p, q)

1: ϑ := an $O(\Delta^2)$-coloring of G /* using set-systems */
2: ψ :=Defective-Col-LEG(G, ϑ, p, q) /* set-system free */
3: return ψ

Set $p = \Delta^\epsilon, q = \Delta^{3\epsilon}$, for an arbitrarily small constant $\epsilon > 0$. By Lemma 2.2 (3), Procedure Defective-

Col-SET invoked with these parameters computes an $O(\Delta/p)$-defective p^2-coloring of G.

Next we sketch a procedure devised in [10] for computing a legal $O(\Delta^{5/4})$-coloring from a legal $O(\Delta^2)$-coloring in $O(\Delta^{3/4})$ time. This procedure is called Procedure Trade-Col-SL. (The properties of this procedure in its general form are summarized in Lemma 2.2 (2). In the current discussion we fix the parameter t to be equal to $\Delta^{1/4}$.) Procedure Trade-Col-SL accepts as input a graph G and a legal $O(\Delta^2)$-coloring ϑ of G. The procedure proceeds as follows. First, it computes an $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring of the input graph using Procedure Defective-Col-LEG. (See Lemma 2.2 (3).) To this end we set $p = \Delta^{1/4}, q = \Delta^{3/4}$. (Normally, it is preceded by a step in which an $O(\Delta^2)$-coloring is computed via Linial's algorithm, i.e., using set systems. In the current version of the procedure, this coloring is assumed to be provided as a part of the input.) The defective coloring returned by the invocation of Procedure Defective-Col-LEG induces a *vertex* partition into $O(\Delta^{1/2})$ subgraphs such that each subgraph has maximum degree $O(\Delta^{3/4})$. Next, all subgraphs are legally colored with distinct palettes of size $O(\Delta^{3/4})$ for each subgraph, in parallel. Then these colorings are combined into a unified legal $O(\Delta^{5/4})$-coloring. This completes the description of the Procedure. The pseudocode of Procedure Trade-Col-SL is provided in Algorithm 4. Similarly to the previous example, the computation is divided into stages that either involve set-systems or are set-systems free.

Algorithm 4 Procedure Trade-Col-SL (G , ϑ)

1: ψ := Defective-Col-LEG (G, ϑ, $p := \Delta^{1/4}$, $q := \Delta^{3/4}$) /* set-system free; see Lemma 2.2 (3) */
/* ψ is an $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring of G */
/* ψ induces a vertex partition into $O(\Delta^{1/2})$ subgraphs, each with max. degree $O(\Delta^{3/4})$ */
2: **for** each subgraph G_i induced by color classes of ψ, in parallel **do**
3: φ_i := color G_i with $O(\Delta^{3/2})$ colors using Linial's algorithm [37] /* using set-systems */
4: φ'_i := Delta-Col-SL(G_i, φ_i) /* using set-systems; see Lemma 2.2 (1) */
/* φ'_i is an $O(\Delta^{3/4})$-coloring of G_i */
5: **end for**
6: **for** $i = 1, 2, \ldots$ in parallel **do**
7: combine all colorings φ'_i into a unified legal $O(\Delta^{5/4})$-coloring φ of G /* set-system free */
8: **end for**
9: return φ

Next, we turn to describing our new set-system free algorithm for computing an $O(\Delta^{5/4})$-coloring from scratch. Our algorithm employs the method Generic-Merge, that was described in Sect. 3. In Sect. 4.1 the KW Procedure was used as an instantiation for Procedure Reduce in the method Generic-Merge. This time a different procedure is used as an instantiation for Procedure Reduce. This procedure reduces the number of colors from $c^2 \cdot \Delta^{5/2}$ to $c \cdot \Delta^{5/4}$, for a positive constants c. Its running time is $O(\Delta^{3/4} \log \Delta) = o(\Delta/\log \Delta)$. Consequently, the $\lceil \log \Delta \rceil$ phases of Generic-Merge require overall time of $o(\Delta)$. For $i = 1, 2, \ldots, \lceil \log \Delta \rceil$, the colorings of subgraphs in the partition of phase i employ at most $(c \cdot \Delta^{5/4})$ colors each. In phase i, pairs of $(c \cdot \Delta^{5/4})$-colored subgraphs are merged into $(c^2 \cdot \Delta^{5/2})$-colored subgraphs, and the number of colors is then reduced back to $(c \cdot \Delta^{5/4})$ in each subgraph. Once the method Generic-Merge terminates, the input graph is colored with $(c \cdot \Delta^{5/4})$ colors.

We use a modified version of Procedure Trade-Col-SL (Algorithm 4), as an instantiation for Procedure Reduce (see Property 3.2). Given an $O(\Delta^2)$-coloring as input, Procedure Trade-Col-SL computes an $O(\Delta^{5/4})$-coloring in $O(\Delta^{3/4})$ time. Some of its steps employ set-systems. (See Algorithm 4). Consequently, the original Procedure Trade-Col-SL cannot be used. We perform two modifications in the procedure. First, the steps that employ set-systems are replaced by analogous set-system free steps. Second, we show that the procedure computes an $O(\Delta^{5/4})$-coloring from an $O(\Delta^k)$-coloring for any constant $k \geq 2$, not only $k = 2$. We henceforth refer to the modified procedure as *Procedure Mod-Trade-LEG*.

Procedure Trade-Col-SL employs set-systems for computing $O(\Delta^{3/2})$-colorings of subgraphs G_i (line 3 of Algorithm 4). In addition, it employs set-systems in the invocation of Procedure Delta-Col-SL for computing legal colorings of subgraphs with linear number of colors (and in linear time) in the degree of the subgraphs (line 4 of Algorithm 4). In Procedure Mod-Trade-LEG we omit the step of computing $O(\Delta^{3/2})$-coloring of subgraphs G_i. Instead of this step, φ_i is set as the initial coloring ϑ, for all i. For each $i = 1, 2, \ldots, O(\Delta^{1/2})$, let $\Delta_i = \Theta(\Delta^{3/4})$ be an upper bound on the maximum degree $\Delta(G_i)$ of the graph G_i. Since ϑ is an $O(\Delta^2)$-coloring, it is also an $O(\Delta_i^{8/3})$-coloring of G_i. Next, we replace the step of coloring G_i using Procedure Delta-Col-SL with an invocation of the KW iterative procedure, which is set-systems free. This procedure can start (see Lemma 2.1) with an arbitrarily large number of colors, as long as it is polynomial in the maximum degree of the underlying graph. However, as a result, the running time of procedure Mod-Trade-LEG grows by a multiplicative factor of $\log \Delta$, and becomes $O(\Delta^{3/4} \log \Delta)$.

The original Procedure Trade-Col-SL accepts as input a $(c' \cdot \Delta^2)$-coloring ϑ, for a positive constant c', and reduces it into an $O(\Delta^{5/4})$-coloring. Once line 3 of Algorithm 4 is skipped,[5] the coloring ϑ is used only in one step of Procedure Trade-Col-SL, specifically, in the step that computes

[5] Line 3 of Algorithm 4 invokes the algorithm of Linial. The latter algorithm employs the coloring ϑ for a faster computation of the colorings φ_i.

the $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring (line 1 of Algorithm 4). In this step a procedure called Procedure Defective-Col-LEG is invoked with two input parameters $p = \Delta^{1/4}$ and $q = \Delta^{\epsilon} \cdot p^2$, for an arbitrary small positive constant $\epsilon \leq 1/4$. Let t be the number of colors used by the legal coloring ϑ. Procedure Defective-Col-LEG employs the coloring ϑ to compute a $(\frac{\log t}{\log(q/p^2)} \cdot \Delta/p)$-defective p^2-coloring in time $(\frac{\log t}{\log(q/p^2)}) \cdot O(q)$. (See Lemma 2.2 (3).) If $t = \Delta^{O(1)}$, then Procedure Defective-Col-LEG computes an $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring in time $O(q) = O(\Delta^{\epsilon} \cdot p^2) = O(\Delta^{1/2+\epsilon})$. We stress that this result holds even when starting with a legal coloring ϑ that employs $t = \Delta^{O(1)}$ colors, and it is not limited to $t = O(\Delta^2)$.

The modified Procedure Mod-Trade-LEG proceeds as follows. It accepts as input a $\Delta^{O(1)}$-coloring ϑ. First, it computes an $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring ψ of the input graph in time $O(\Delta^{1/2+\epsilon})$ by Procedure Defective-Col-LEG as was described above. Next, the subgraphs $G_1, G_2, \ldots$ induced by color classes of ψ are legally colored with distinct palettes of size $O(\Delta^{3/4})$ each, using the KW iterative procedure in parallel on all subgraphs, in time $O(\Delta^{3/4} \cdot \log(\frac{\Delta^{O(1)}}{\Delta^{3/4}})) = O(\Delta^{3/4} \log \Delta)$. (Observe that the KW iterative procedure invoked on a subgraph G_i needs an initial coloring ϑ_i to start working. Here we restrict the coloring ϑ of the entire graph G to the vertex set V_i of the subgraph G_i. The resulting restricted coloring is called ϑ_i, and is employed by the KW iterative procedure as an initial coloring.) Then the colorings produced by the invocations of the KW iterative procedure are combined into a unified legal $O(\Delta^{5/4})$ coloring. The combining step requires no communication whatsoever. This completes the description of the procedure. Its pseudocode is provided below. Its properties are given in the next lemma.

Lemma 4.2 *Procedure Mod-Trade-LEG invoked with a $\Delta^{O(1)}$-coloring ϑ as input computes an $O(\Delta^{5/4})$-coloring φ in time $O(\Delta^{3/4} \log \Delta)$. Specifically, if ϑ is a $(c' \cdot \Delta^k)$-coloring, for constants $c' > 0, k \geq 2$, then φ employs at most $(\frac{\log(c' \cdot \Delta^k)}{\log \Delta^{\epsilon}} \cdot \Delta^{5/4}) = O(\frac{k}{\epsilon} \cdot \Delta^{5/4})$ colors.*

Algorithm 5 Procedure Mod-Trade-LEG (G , ϑ)

1: ψ :=Defective-Col-LEG (G, ϑ, $\Delta^{1/4}$, $\Delta^{3/4}$)
/* ψ is an $O(\Delta^{3/4})$-defective $O(\Delta^{1/2})$-coloring */
2: **for** each subgraph G_i induced by ψ, in parallel **do**
3: φ'_i := color G_i with $O(\Delta^{3/4})$ colors using the KW iterative procedure
/* use the restriction ϑ_i of the coloring ϑ to the vertex set V_i of G_i as initial coloring */
4: **end for**
5: **for** $i = 1, 2, \ldots,$ in parallel **do**
6: combine all colorings φ'_i into a unified legal $O(\Delta^{5/4})$-coloring φ of G
7: **end for**
8: return φ

Suppose that Procedure Mod-Trade-LEG is invoked with a $(c^2 \cdot \Delta^{5/2})$-coloring ϑ as input, for a positive constant c to be determined later. Then it computes a $(\frac{\log(c^2 \cdot \Delta^{5/2})}{\log \Delta^{\epsilon}} \cdot \Delta^{5/4})$-coloring φ, for an arbitrary positive constant $\epsilon \leq 1/4$. For any constant c such that $c \geq \frac{2 \log c + 5/2}{\epsilon}$ it holds that $\frac{\log(c^2 \cdot \Delta^{5/2})}{\log \Delta^{\epsilon}} \leq c$. (To satisfy the condition set $\epsilon = 1/8$, and c to be a sufficiently large constant.) Consequently, the resulting coloring φ is a $\lfloor c \cdot \Delta^{5/4} \rfloor$-coloring.

Next, we present a set-system free procedure, called Procedure Poly-Col, that computes an $O(\Delta^{5/4})$-coloring of the input graph *from scratch*, in time $\tilde{O}(\Delta^{3/4}) + \log^* n$. (This is in contrast to Procedure Defective-Col-LEG that accepts as input a legal $O(\Delta^2)$-coloring ϑ). Procedure Poly-Col is closely related to Procedure Simple-Col (Algorithm 2). The main difference is that in step 6 Procedure Poly-Col invokes the method Generic-Merge with Procedure Mod-Trade-LEG as an instantiation for Procedure Reduce instead of the KW Procedure. In addition, the variables $c_1, c_2, \ldots, c_{\Delta}$, and d are set to $\lfloor c \cdot \Delta^{5/4} \rfloor$ instead of $\Delta + 1$, where c is a constant as above. The pseudocode of the procedure is given below. Its properties are summarized in the next lemma.

Algorithm 6 Procedure Poly-Col(G)

1: $\{G_1, G_2, ..., G_{\Delta}\}$:= Δ-Forest-Decomposition(G)
/* using PR Procedure */
2: **for** $i = 1, 2, ..., \Delta$ in parallel **do**
3: $c_i := \lfloor c \cdot \Delta^{5/4} \rfloor$; φ_i := 3-color(G_i)
/* using GP Algorithm; each G_i is a forest */
4: **end for**
5: $d := \lfloor c \cdot \Delta^{5/4} \rfloor$
6: Generic-Merge(P,G_1, G_2, ..., G_{Δ}, φ_1, φ_2, ..., φ_{Δ}, c_1, c_2, ..., c_{Δ}, Procedure Mod-Trade-LEG, d)

Lemma 4.3 *Procedure Poly-Col computes from scratch a legal coloring of G that employs at most $c \cdot \Delta^{5/4} = O(\Delta^{5/4})$ colors. Its running time is $O(\Delta^{3/4} \log^2 \Delta + \log^* n)$. Moreover, Procedure Poly-Col is set-system free.*

Proof The correctness of Procedure Poly-Col follows directly from Lemma 3.3. Next, we prove that the running time of Procedure Poly-Col is $O(\Delta^{3/4} \log^2 \Delta + \log^* n)$. Steps 1–5 of Procedure Poly-Col require $O(\log^* n)$ time. In step 6 the method Generic-Merge is invoked in conjunction with Procedure Mod-Trade-LEG. In each of its $\lceil \log \Delta \rceil$ phases, the method Generic-Merge invokes Procedure Mod-Trade-LEG that requires $O(\Delta^{3/4} \log \Delta)$ time. (See Lemma 4.2). Therefore, the overall running time is $O(\Delta^{3/4} \log^2 \Delta + \log^* n)$. □

4.3 Procedure Fast-Col

In this section we devise a set-system free procedure for computing a $(\Delta + 1)$-coloring in $O(\Delta + \log^* n)$ time. In [10] the

authors of the current paper devised a procedure, called *Procedure Delta-Color* (henceforth, *Procedure Delta-Col-SL*), that accepts as input a graph G and a γ-coloring ϑ, $\gamma = O(\Delta^2)$, and computes a $(\Delta+1)$-coloring of G in $O(\Delta)$ time. (See Lemma 2.2). However, Procedure Delta-Col-SL employs set-systems. Next, we overview Procedure Delta-Col-SL from [10]. This procedure works as follows. If the number of colors in the coloring it accepts as input is $\gamma = O(\Delta)$, then a $(\Delta+1)$-coloring of G is computed directly from the γ-coloring ϑ using the KW iterative procedure within $O(\Delta)$ time. Otherwise, the graph G is partitioned into vertex-disjoint subgraphs with maximum degrees $d < \Delta$, by invoking Procedure Defective-Col-LEG. Next, for each subgraph, an $O(d^2)$-coloring is computed using set-systems, by Linial's algorithm [37]. Then Procedure Delta-Col-SL is invoked recursively on each of the subgraphs. As a result we obtain a $(d+1)$-coloring for each subgraph. (The recursion depth is $\lfloor \log^* \Delta \rfloor$. For $j = 1, 2, \ldots, \lfloor \log^* \Delta \rfloor$, in recursion level $\lfloor \log^* \Delta \rfloor - j$ it holds that $d = d_j = \Theta(\Delta / \Pi_{i=j}^{\log^* \Delta}(\log^{(i)} \Delta))$, i.e., $d_j = \Omega(\Delta^{1-\epsilon})$ for any constant $\epsilon > 0$.) These colorings are then merged into a unified legal $(\Delta+1)$-coloring of the input graph. This completes the description of Procedure Delta-Col-SL. Its pseudocode is provided for completeness in Algoritm 7. The running time of a recursion level j, $j = 1, 2, \ldots, \lfloor \log^* \Delta \rfloor$, is $O(d_j \cdot \log^{(j)} \Delta)$. Consequently, the overall running time is $O(\sum_{j=1}^{\log^* \Delta}(d_j \cdot \log^{(j)} \Delta)) = O(\Delta)$.

The only step in Procedure Delta-Col-SL that employs set-systems is the step that computes $O(d^2)$-colorings. Specifically, let $d_j = c \cdot \Delta / \Pi_{i=j}^{\log^* \Delta}(\log^{(i)} \Delta)$, for some fixed positive constant c. For $j = 1, 2, \ldots, \lfloor \log^* \Delta \rfloor$, Procedure Delta-Col-SL computes $O(d_j^2)$-colorings of subgraphs of degree $O(d_j)$. To this end it employs the algorithm of Linial [37]. We argue that if Procedure Delta-Col-SL accepts an $O(\Delta^{5/4})$-coloring instead of an $O(\Delta^2)$-coloring, then employing set systems is no longer required. Observe that as $d_j = \Omega(\Delta^{1-\epsilon})$, for any constant $\epsilon > 0$, it follows that for a sufficiently large Δ, $\Delta^{5/4} \leq d_j^2$, for all $j = 1, 2, \ldots, \lfloor \log^* \Delta \rfloor$. Hence an $O(\Delta^{5/4})$-coloring is in particular an $O(d_j^2)$-coloring for all $j = 1, 2, \ldots, \lfloor \log^* \Delta \rfloor$. Therefore, invoking Procedure Delta-Col-SL with an $O(\Delta^{5/4})$-coloring as input instead of an $O(\Delta^2)$-coloring eliminates the need of computing $O(d_j^2)$-colorings of subgraphs. Indeed, for a subgraph H of degree d_j, $j \in \{1, 2, \ldots, \lfloor \log^* \Delta \rfloor\}$, the input coloring is already an $O(d_j^2)$-coloring of H.

To summarize, we obtained a variant of Procedure Delta-Col-SL, to which we will refer as Procedure Mod-Delta-LEG. This procedure accepts as input a graph G of maximum degree Δ, and an $O(\Delta^{5/4})$-coloring ϑ for G. The procedure returns a $(\Delta+1)$-coloring, and it does so within $O(\Delta)$ time.

Algorithm 7 Procedure Delta-Col-SL(G, Λ, i, ϑ)

Input: a graph G with maximum degree $\Lambda = \Delta$, a parameter i that determines the depth of the recursion, and is set as $i = \lfloor \log^* \Delta \rfloor$, and an initial coloring ϑ of G.
Output: a legal $(\Delta+1)$-coloring of G.

1: **if** i = 1 **then**
2: compute a $(\Lambda+1)$-coloring of G from ϑ using the KW iterative procedure
3: **else**
4: $k := \lfloor \log^{(i-1)} \Lambda \rfloor$
5: $d := \lfloor \Lambda / k \rfloor$
6: $\varphi :=$ Procedure Defective-Col-LEG (G, ϑ, $p := k$, $q := \lfloor \Lambda^{\epsilon} \rfloor$)
7: let V_j, $j = 1, 2, \ldots, k^2$, denote the set of vertices such that $\varphi(v) = j$
8: **for** $j = 1, 2, \ldots, k^2$, in parallel **do**
9: $\varphi_j :=$ compute an $O(d^2)$-coloring of $G(V_j)$ /* using set-systems */
10: $\varphi'_j :=$ invoke Procedure Delta-Col-SL($G(V_j)$, d, $i-1$, φ_j) recursively
11: **for** each $v \in G(V_j)$, in parallel **do**
12: $\vartheta(v) := \varphi'_j(v) + (d+1)(j-1)$
13: **end for**
14: **end for**
15: compute a $(\Lambda+1)$-coloring of G from ϑ using the KW iterative procedure
16: **end if**

(Observe that the running time of Procedure Mod-Delta-LEG is not greater than the running time of Procedure Delta-Col-SL when invoked with an $O(\Delta^{5/4})$-coloring as input. The two procedures perform exactly the same steps, except that Procedure Mod-Delta-LEG skips the invocation of the algorithm of Linial).

To complete the algorithm it is only left to combine Procedure Poly-Col (that computes an $O(\Delta^{5/4})$-coloring from scratch) with Procedure Mod-Delta-LEG that reduces the number of colors to $\Delta+1$. The resulting procedure will be referred to as Procedure Fast-Col. It accepts as input a graph G, and performs two steps. In the first step it computes an $O(\Delta^{5/4})$-coloring ϑ using Procedure Poly-Col. In the second step it invokes the set-system free variant (Procedure Mod-Delta-LEG) of Procedure Delta-Col-SL with the coloring ϑ as input. The pseudocode of Procedure Fast-Col is provided below.

Algorithm 8 Procedure Fast-Col(G)

1: $\vartheta :=$ Poly-Col(G)
2: $\psi :=$ Mod-Delta-LEG(G, ϑ)
3: return ψ

This procedure outputs a $(\Delta+1)$-coloring. The running time of Procedure Fast-Col is the sum of the running times of Procedure Poly-Col and Procedure Mod-Delta-LEG. The former is, by Lemma 4.3, $O(\Delta^{3/4} \log^2 \Delta) + \log^* n$, and the latter is $O(\Delta)$. Hence the overall running time of Procedure Fast-Col is $O(\Delta) + \log^* n$.

Theorem 4.4 *Procedure Fast-Col computes a $(\Delta+1)$-coloring of the input graph G in $O(\Delta+\log^* n)$ time. Moreover, this computation is set-system free.*

5 A tradeoff

In this section we devise a set-system free algorithm that provides a tradeoff of $O(\Delta \cdot t)$-coloring within $O(\Delta/t + \log^* n)$ time, for every t, $1 \leq t \leq \Delta^{1-\epsilon}$, for some constant arbitrarily small $\epsilon > 0$. We argue that Procedure Mod-Trade-LEG can be extended to compute in a set-system free manner an $O(\Delta \cdot t)$-coloring from scratch in time $O((\Delta/t)\log\Delta + \log^* n)$ for any $1 < t \leq \Delta^{1-\epsilon}$, for an arbitrarily small positive constant ϵ. (By Lemma 4.2, the current version of Procedure Mod-Trade-LEG does this for the somewhat arbitrary value $t = \Delta^{1/4}$, and requires a legal $\Delta^{O(1)}$-coloring ϑ as input.) We remark that the original Procedure Trade-Col-SL (see Algorithm 4; this procedure employs set-systems) can be invoked with any parameter t, $1 \leq t \leq \Delta^{1-\epsilon}$, not only $t = \Delta^{1/4}$. (See [10], Corollary 4.9).

As a first step, we show that defective colorings can be computed, in a set-system free manner, much faster than stated in Lemma 2.2 (3). The argument that shows it is implicit in [10]. Below we sketch it for the sake of completeness. By Lemma 2.2 (3), if we set $p = \Delta^{\epsilon}, q = p^3$, for an arbitrarily small constant $\epsilon, 0 < \epsilon < 1$, then, given a $\Delta^{O(1)}$-coloring ϑ as input, Procedure Defective-Col-LEG computes an $O(\Delta^{1-\epsilon})$-defective $\Delta^{2\epsilon}$-coloring of the input graph. The color classes of this coloring induce a partition into $p^2 = \Delta^{2\epsilon}$ subgraphs $G_1, G_2, \ldots, G_{p^2}$ with maximum degree $\bar{\Delta} = O(\Delta^{1-\epsilon})$ in each subgraph. Since $(1-\epsilon) = \Omega(1)$, the coloring ϑ is, in particular, a $\bar{\Delta}^{O(1)}$-coloring of G_i, for $i = 1, 2, \ldots, p^2$. Therefore, we can invoke Procedure Defective-Col-LEG again to compute an $O(\Delta^{1-2\epsilon})$- defective $\Delta^{2\epsilon}$-coloring ϑ_i in each G_i. If distinct palettes are used for each subgraph G_i, then the coloring $\vartheta_1, \vartheta_2, \ldots, \vartheta_{p^2}$ can be combined into a unified $O(\Delta^{1-2\epsilon})$-defective $\Delta^{4\epsilon}$-coloring of G. The overall running time to obtain such a coloring is $O(\Delta^{3\epsilon})$, rather than $O(\Delta^{6\epsilon})$ that would be achieved if we used Lemma 2.2 (3) directly with the parameters $p = \Delta^{2\epsilon}, q = p^3$.

In general, for any $1 \leq t \leq \Delta^{1-\eta}$, an $O(\Delta/t)$-defective t^2-coloring can be computed from a $\Delta^{O(1)}$-coloring in $O(\Delta^{\epsilon})$ time, for arbitrarily small constants $\epsilon, \eta > 0$. This coloring is obtained by performing $O(\frac{\log t}{\log(\Delta^{\epsilon})}) = O(\frac{\log t}{\log \Delta}) = O(1)$ iterations, starting from an initial Δ-defective 1-coloring, and using the input $\Delta^{O(1)}$-coloring ϑ. (A Δ-defective 1-coloring is obtained by simply coloring all the vertices of the input graph with the same color.) In iteration i, for $i = 1, 2, \ldots, O(1)$, an $O(\Delta^{1-i\cdot\epsilon})$-defective $O(\Delta^{i\cdot 2\epsilon})$-coloring φ_i is obtained from an $O(\Delta^{1-(i-1)\cdot\epsilon})$-defective $O(\Delta^{(i-1)\cdot 2\epsilon})$-coloring φ_{i-1} and the coloring ϑ. Observe that the maximum degree of the subgraphs induced by the color classes of φ_{i-1} is $\Delta_{i-1} = O(\Delta^{1-(i-1)\cdot\epsilon})$. Observe also that ϑ is a $\Delta_i^{O(1)}$-coloring. In iteration i, we compute an $O(\frac{\Delta_{i-1}}{\Delta^{\epsilon}})$-defective $\Delta^{2\epsilon}$-coloring in each subgraph using distinct palettes. These coloring are combined into a unified $O(\Delta^{1-i\cdot\epsilon})$-defective $O(\Delta^{i\cdot 2\epsilon})$-coloring φ_i of the input graph. The running time of each iteration is $O(\Delta^{\epsilon'})$, for $\epsilon' = 3\epsilon$. Since $\epsilon > 0$ is an arbitrarily small positive constant, so is ϵ' as well. This iterative process is terminated once $\Delta^{i\cdot\epsilon} \geq t$. (In the last iteration it is possible to set a smaller value for p, to achieve the exact threshold t). The properties of the described algorithm are summarized in the following Lemma.

Lemma 5.1 *Let η be an arbitrarily small positive constant, and t be a parameter such that $1 \leq t \leq \Delta^{1-\eta}$. Given a graph G with an initial $\Delta^{O(1)}$-coloring, for any arbitrarily small constant $\epsilon > 0$, one can compute an $O(\Delta/t)$-defective t^2-coloring in $O(\Delta^{\epsilon})$ time. Moreover, this computation is set-system free.*

Next, we describe the required changes in Procedure Mod-Trade-LEG. The new procedure accepts as input an additional parameter t, which is taken into account during the execution of the Procedure. In addition, step 1 of the procedure is replaced with the algorithm given in Lemma 5.1. The pseudocode of the new procedure is given below. (It is instructive to compare it with Algorithm 5, in which a similar computation is done for $t = \Delta^{1/4}$.) The next theorem summarizes its properties.

Algorithm 9 Procedure New-Trade-LEG (G , ϑ , t)

1: ψ := compute an $O(\Delta/t)$-defective t^2-coloring using Lemma 5.1
2: **for** each subgraph G_i induced by ψ, in parallel **do**
3: φ'_i := color G_i with $O(\Delta/t)$ colors using the KW iterative procedure
/* use the restriction ϑ_i of the coloring ϑ to the vertex set V_i of G_i as an initial coloring */
4: **end for**
5: **for** $i = 1, 2, ...,$ in parallel **do**
6: combine all colorings φ'_i into a unified legal $O(\Delta \cdot t)$-coloring φ of G
7: **end for**
8: return φ

Theorem 5.2 *Let G be an input graph with a $\Delta^{O(1)}$-coloring ϑ, and let t be a parameter such that $1 < t \leq \Delta^{1-\epsilon}$, for an arbitrarily small constant $\epsilon > 0$. Procedure New-Trade-LEG computes an $O(\Delta \cdot t)$-coloring of G in $O(\Delta/t \cdot \log\Delta)$ time.*

Given Lemma 5.1, the proof of Theorem 5.2 is very similar to the proof of Lemma 4.2 (that analyzes the special case $t = \Delta^{1/4}$).

Moreover, Procedure Mod-Trade-LEG can be further improved to produce an $O(\Delta \cdot t)$-coloring in time $O(\Delta/t + \log^* n)$, i.e., the slack factor of $\log \Delta$ in its running time can be eliminated. To this end, replace the invocation of the KW procedure in Algorithm 5, line 3, with an analogous invocation of Procedure Fast-Col. (Recall that by Theorem 4.4, Procedure Fast-Col requires $O(\Delta + \log^* n)$ time, rather than $O(\Delta \log \Delta)$ that the KW procedure requires.) However, Procedure Mod-Trade-LEG accepts a legal $\Delta^{O(1)}$-coloring as input. Next, we show that one can convert it into a similar procedure that operates *from scratch*. The resulting procedure will be referred to as *Procedure New-Trade*. Using the extended version of Procedure Mod-Trade-LEG, the Procedure Poly-Col can also be extended to compute an $O(\Delta \cdot t)$-coloring from scratch in time $O((\Delta/t) \cdot \log^2 \Delta + \log^* n)$ for any $1 < t \leq \Delta^{1-\epsilon}$, for an arbitrarily small positive constant ϵ. (See Lemma 4.3.) Consequently, one can compute from scratch an $O(\Delta \cdot t)$ coloring in time $O((\Delta/t) + \log^* n)$ by executing the following two steps. In the first step compute an $O(\Delta \log^2 \Delta \cdot t)$-coloring ϑ using the extended version of Procedure Poly-Col in time $O((\Delta/t) + \log^* n)$. In the second step invoke the extended version of Procedure Mod-Trade-LEG, with ϑ as input. This extended version produces an $O(\Delta \cdot t)$-coloring in time $O((\Delta/t) + \log^* n)$. Overall, we obtain an $O(\Delta \cdot t)$-coloring from scratch in time $O((\Delta/t) + \log^* n)$. This tradeoff exactly matches the tradeoff shown in [10] and in [32], and it is obtained by a set-system free computation. (However, the tradeoff of [10] and [32] is obtained using set-systems.) We summarize this discussion in the following corollary.

Corollary 5.3 *For any arbitrarily small constant $\epsilon > 0$, and a parameter t, $1 < t \leq \Delta^{1-\epsilon}$, Procedure New-Trade computes an $O(\Delta \cdot t)$-coloring in $O((\Delta/t) + \log^* n)$ time. Moreover, this computation is set-system free.*

6 Applications

Vertex coloring is used by algorithms for various network tasks such as computing maximal independent set (henceforth, MIS), edge coloring, maximal matching and others. Since the previously known methods for efficiently computing vertex colorings involve set systems, all these algorithms use set systems as well. Therefore, eliminating the use of set systems from vertex coloring algorithms gives rise to set-system free algorithms for many other fundamental network tasks. Moreover, there is a wide range of algorithms that do not employ coloring directly, but rather invoke some other symmetry breaking procedures during their execution. These symmetry breaking procedures are often variations of vertex coloring, MIS, edge coloring, and maximal matching. For example, some algorithms for computing minimum spanning tree [26,36] perform MIS computations during execution. In order to eliminate the use of set-systems from algorithms that empoy MIS computations one needs first to make MIS algorithms set-systems free.

Currently, the state-of-the-art algorithm for computing an MIS on graphs with small maximum degree ($\Delta = o(\log n)$) employs a reduction that accepts a k-coloring, for a positive integer k, and transforms it into an MIS in k rounds. (For the description of this simple reduction, see, e.g., [46], Chapter 8). In conjunction with a $(\Delta+1)$-coloring algorithm of [10], this reduction enables us to compute an MIS from scratch in $O(\Delta + \log^* n)$ time. The algorithm involves the use of set systems. Our new results (Theorem 4.4) imply that this computation can be made set-system free. This fact is given in the following corollary.

Corollary 6.1 *An MIS can be computed in $O(\Delta + \log^* n)$ time using a set-system free algorithm.*

For certain families of graphs more efficient MIS algorithms are known. In particular, for graphs of *bounded arboricity* $a(G) = o(\sqrt{\log n})$ an MIS can be computed in sublogarithmic time [9]. Next, we describe a set-system free algorithm for this task. The algorithm will be referred to as *Procedure New-MIS*. The computation begins by partitioning the vertex set of the graph into $\ell = o(\log n)$ subsets $H_1, H_2, \ldots, H_\ell$, each of maximum degree $A = o(\sqrt{\log n})$. Moreover, this partition has the property that for every index i, $1 \leq i \leq \ell$, and every vertex $v \in H_i$, the number of neighbors that v has in $\bigcup_{j=i}^{\ell} H_j$ is at most A. Next, for $i = 1, 2, \ldots, \ell$ in parallel, an $(A+1)$-coloring φ_i is computed for H_i. Then an additional auxiliary (not necessarily legal) $(A+1)$-coloring φ' is computed. Each vertex $v \in H_\ell$ sets its φ' color to be equal to its φ_ℓ-color, i.e., sets $\varphi'(v) := \varphi_\ell(v)$. For some index i, $1 < i \leq \ell$, after all vertices of $\bigcup_{j=i}^{\ell} H_j$ determined their φ'-colors, vertices of H_{i-1} determine their φ'-colors in parallel in the following way. Each vertex $v \in H_{i-1}$ selects a color $\varphi'(v)$ from $\{1, 2, \ldots, A+1\}$ that is not used by any of its neighbors in $\bigcup_{j=i}^{\ell} H_j$. In other words, $\varphi'(v) \in \{1, 2, \ldots, A+1\} \setminus \{\varphi(u) \mid u \in \bigcup_{j=i}^{\ell} H_j, (u, v) \in E\}$. The resulting coloring φ' has the property that for any $u \in H_i, v \in H_j, i \neq j$, it holds that $\varphi'(u) \neq \varphi'(v)$. (On the other hand, two neighboring vertices u, w that belong to the same set H_i may end up getting the same φ'-color.) The computation of φ' requires ℓ rounds. Then, for all $i = 1, 2, \ldots, \ell$, each vertex $v \in H_i$ selects its final color $\varphi(v)$ as an ordered pair $\langle \varphi_i(v), \varphi'(v) \rangle$. The coloring φ is a legal $(A+1)^2$-coloring. This coloring is transformed into an MIS within $(A+1)^2$ rounds using the aforementioned standard color reduction. (See, e.g., [46], Chapter 8).

The only step in the above algorithm that involves set systems is computing an $O(A)$-coloring of the graphs $H_1, H_2, \ldots, H_\ell$. (The original algorithm of [9] for this task is

slightly different, and it involves also an invocation of Procedure Arb-Linial. The latter procedure is based on set-systems. On the other hand, in the variant of the algorithm that we described above we bypass the invocation of Procedure Arb-Linial.) By replacing the set-system algorithm that computes the colorings $\varphi_1, \varphi_2, \ldots \varphi_\ell$ with our Procedure Fast-Col, the MIS computation becomes set-system free.

Next, we analyze the running time of this algorithm. For $A = a \cdot q, q > 2$, computing the H-partition $H_1, H_2, \ldots, H_\ell$ requires $\ell = O(\frac{\log n}{\log q})$ time. Computing the colorings $\varphi_1, \varphi_2, \ldots, \varphi_\ell$ is done within $O(a \cdot q + \log^* n)$ time. computing the auxiliary coloring φ' requires additional $\ell = O(\frac{\log n}{\log q})$ rounds. Finally, given an $O(A^2) = O(a^2 \cdot q^2)$-coloring, an MIS (and a $(\Delta+1)$-coloring) can be computed within additional $O(a^2 \cdot q^2)$ time. Hence the overall running time is $O(\frac{\log n}{\log q})$. For $a \leq \log^{1/2-\epsilon}$, for some constant $\epsilon, 0 < \epsilon < 1/2$, we set $q = \log^{\epsilon/3} n$, and obtain running time of $O(\frac{\log n}{\log \log n})$. More generally, suppose that $a = o(\sqrt{\log n})$, i.e., $a = \frac{\sqrt{\log n}}{f(n)}$, for some $f(n) = \omega(1)$. Then we set $q = (f(n))^{1/3}$, and obtain running time of $O(\frac{\log n}{\log \log n}) = o(\log n)$. We summarize this discussion in the following corollary.

Corollary 6.2 *For the family of graphs with bounded arboricity $a(G) = o(\sqrt{\log n})$, an MIS and $(\Delta+1)$-coloring can be computed in sublogarithmic time using a set-system free algorithm. Specifically, if $a(G) \leq \log^{1/2-\epsilon} n$, for some constant $\epsilon, 0 < \epsilon < 1/2$, then the running time of Procedure New-MIS is $O(\frac{\log n}{\log \log n})$.*

7 Conclusion

We have presented a variety of combinatorial distributed algorithms whose running time matches that of the state-of-the art algebraic ones. Our algorithms solve the coloring and MIS problems without using heavy algebraic computations. In addition, they benefit from improved memory complexity and local computation complexity. Improving the algorithms further is an important open problem. Obtaining combinatorial algorithms that outperform algebraic ones (in terms of running time) seems very challenging since combinatorial algorithms are more restrictive. However, such an improvement would be a significant achievment both from theoretical and practical points of view. Another interesting research direction is investigating additional problems for which the state-of-the-art algorithms employ heavy structures, and devising combinatorial counterparts. One such example is the work of [8] that presents a combinatorial algorithm for a wireless synchronization problem. This work improves upon a previous result that is based on expander constructions [15]. Expanders are common structure in the distributed setting that are used in numerous algorithms for various problems. Eliminating the use of expanders from additional algorithms would be very interesting.

Acknowledgments We are grateful to an anonymous reviewer of the Distributed Computing Journal, whose numerous comments helped us to improve the presentation in this paper.

References

1. Aingworth, D., Chekuri, C., Indyk, P., Motwani, R.: Fast estimation of diameter and shortest paths (without matrix multiplication). SIAM J. Comput. **28**(4), 1167–1181 (1999)
2. Ajtai, M.: Recursive construction for 3-regular expanders. Combinatorica **14**(4), 379–416 (1994)
3. Alon, N.: Eigen-values and expanders. Combinatorica **6**(2), 83–96 (1986)
4. Alon, N., Babai, L., Itai, A.: A fast and simple randomized parallel algorithm for the maximal independent set problem. J Algorithms **7**(4), 567–583 (1986)
5. Alon, N., Galil, Z., Margalit, O.: On the exponent of the all pairs shortest path problem. J. Comput. Syst. Sci. **54**(2), 255–262 (1997)
6. Arora, S., Safra, S.: Probabilistic checking of proofs: a new characterization of NP. J. ACM **45**(1), 70–122 (1998)
7. Awerbuch, B., Goldberg, A.V., Luby, M., Plotkin, S.: Network decomposition and locality in distributed computation. In: Proceedings of the 30th IEEE Annual Symposium on Foundations of Computer, Science, pp. 364–369, October 1989
8. Barenboim, L., Dolev, S., Ostrovsky, R.: Deterministic and energy-optimal wireless synchronization. In: Proceedings of the 25th International Symposium on Distributed Computing, pp. 237–251 (2011)
9. Barenboim, L., Elkin, M.: Sublogarithmic distributed MIS algorithm for sparse graphs using Nash-Williams decomposition. In: Proceedings of the 27th ACM Symposium on Principles of, Distributed Computing, pp. 25–34 (2008)
10. Barenboim, L., Elkin, M.: Distributed $(\Delta+1)$-coloring in linear (in Δ) time. In: Proceedings of the 41th ACM Symposium on Theory of Computing, pp. 111–120, 2009. See also http://arXiv.org/abs/0812.1379v2 (2008)
11. Barenboim, L., Elkin, M.: Deterministic distributed vertex coloring in polylogarithmic time. In: Proceedings of the 29th ACM Symposium on Principles of, Distributed Computing, pp. 410–419 (2010)
12. Baswana, S., Kavitha, T.: Faster algorithms for all-pairs approximate shortest paths in undirected graphs. SIAM J. Comput. **39**(7), 2865–2896 (2010)
13. Ben-Aroya, A., Ta-Shma, A.: A combinatorial construction of almost-Ramanujan graphs using the zig-zag product. In: Proceedings of the 40th ACM Symposium on Theory of, Computing, pp. 325–334 (2008)
14. Bilu, Y., Linial, N.: Lifts, discrepancy and nearly optimal spectral gaps. Combinatorica **26**(5), 495–519 (2006)
15. Bradonjić, M., Kohler, E., Ostrovsky, R.: Near-optimal radio use for wireless network synchronization. In: Proceedings of the 5th International Workshop on Algorithmic Aspects of Wireless Sensor, Networks, pp. 15–28 (2009)
16. Cole, R., Vishkin, U.: Deterministic coin tossing with applications to optimal parallel list ranking. Inf. Control **70**(1), 32–53 (1986)
17. Dinur, I.: The PCP theorem by gap amplification. In: Proceedings of the 38th ACM Symposium on Theory of, Computing, pp. 241–250 (2006)

18. Dinur, I., Reingold, O.: Assignment testers: towards a combinatorial proof of the PCP theorem. SIAM J. Comput. **36**(4), 975–1024 (2006)
19. Dor, D., Halperin, S., Zwick, U.: All pairs almost shortest paths. SIAM J. Comput. **29**(5), 1740–1759 (2000)
20. Elkin, M.: Computing almost shortest paths. In: Proceedings of the 20th ACM Symposium on Principles of, Distributed Computing, pp. 53–62 (2001)
21. Elkin, M., Kortsarz, G.: Combinatorial logarithmic approximation algorithm for directed telephone broadcast problem. In: Proceedings of the 34th ACM Symposium on Theory of, Computing, pp. 438–447 (2002)
22. Elkin, M., Kortsarz, G.: Sublogarithmic approximation for telephone multicast: path out of jungle. In: Proceedings of the 14th ACM-Siam Symposium on Discrete Algorithms, pp. 76–85 (2003)
23. Erdős, P., Frankl, P., Füredi, Z.: Families of finite sets in which no set is covered by the union of r others. Israel J. Math. **51**, 79–89 (1985)
24. Feige, U., Goldwasser, S., Lovasz, L., Safra, S., Szegedy, M.: Interactive proofs and the hardness of approximating cliques. J. ACM **43**(2), 268–292 (1996)
25. Galil, Z., Margalit, O.: All pairs shortest distances for graphs with small integer length edges. Inf. Comput. **134**(2), 103–139 (1997)
26. Garay, J.A., Kutten, S., Peleg, D.: A sub-linear timedistributed algorithm for minimum-weight spanning trees. In: Proceedings of the 34th IEEE Annual Symposium on Foundations of Computer, Science, pp. 659–668 (1993)
27. Goldberg, A., Plotkin, S.: Efficient parallel algorithms for $(\Delta+1)$-coloring and maximal independent set problem. In: Proceedings 19th ACM Symposium on Theory of, Computing, pp. 315–324 (1987)
28. Goldberg, A., Plotkin, S., Shannon, G.: Parallel symmetry-breaking in sparse graphs. SIAM J. Discret. Math. **1**(4), 434–446 (1988)
29. Goldreich, O., Safra, S.: A combinatorial consistency lemma with application to proving the PCP theorem. SIAM J. Comput. **29**(4), 1132–1154 (2000)
30. Kale, S., Seshadhri, C.: Combinatorial approximation algorithms for MaxCut using random walks. In: Proceedings of the 2nd Symposium on Innovations in Computer, Science, pp. 367–388 (2011)
31. Kothapalli, K., Scheideler, C., Onus, M., Schindelhauer, C.: Distributed coloring in $O(\sqrt{\log n})$ bit rounds. In: 20th International Parallel and Distributed Processing Symposium (2006)
32. Kuhn, F.: Weak graph colorings: distributed algorithms and applications. In: Proceedings of the 21st ACM Symposium on Parallel Algorithms and Architectures, pp. 138–144 (2009)
33. Kuhn, F., Moscibroda, T., Wattenhofer, R.: What cannot be computed locally! In: Proceedings of the 23rd ACM Symposium on Principles of, Distributed Computing, pp. 300–309 (2004)
34. Kuhn, F., Moscibroda, T., Wattenhofer, R.: Local computation: lower and upper bounds. http://arXiv.org/abs/1011.5470 (2010)
35. Kuhn, F., Wattenhofer, R.: On the complexity of distributed graph coloring. In Proc. of the 25th ACM Symp. on Principles of, Distributed Computing, pp. 7–15 (2006)
36. Kutten, S., Peleg, D.: Fast distributed construction of small k-dominating sets and application. In: Proceedings of the 14th ACM Symposium on Principles of, Distributed Computing, pp. 238–249 (1995)
37. Linial, N.: Locality in distributed graph algorithms. SIAM J. Comput. **21**(1), 193–201 (1992)
38. Lubotzky, A., Philips, R., Sarnak, P.: Ramanujan graphs. Combinatorica **8**, 261–277 (1988)
39. Luby, M.: A simple parallel algorithm for the maximal independent set problem. SIAM J. Comput. **15**, 1036–1053 (1986)
40. Margulis, G.A.: Explicit group-theoretic constructions of combinatorial schemes and their applications in the construction of expanders and concentrators. Problemy Peredaci Informatsii **24**(1), 51–60 (1988)
41. Meir, O.: Combinatorial PCPs with efficient verifiers. In: Proceedings of the 50th Annual IEEE Symposium on Foundations of Computer, Science, pp. 463–471 (2009)
42. Morgenstern, M.: Existence and explicit constructions of q+1 regular Ramanujan graphs for every prime power q. J. Comb. Theory Ser. B **62**(1), 44–62 (1994)
43. Oldham, J.: Combinatorial approximation algorithms for generalized flow problems. In: Proceedings of the 10th ACM-Siam Symposium on Discrete Algorithms, pp. 704–714 (1999)
44. Panconesi, A., Rizzi, R.: Some simple distributed algorithms for sparse networks. Distrib. Comput. **14**(2), 97–100 (2001)
45. Panconesi, A., Srinivasan, A.: On the complexity of distributed network decomposition. J. Algorithms **20**(2), 581–592 (1995)
46. Peleg, D.: Distributed computing: a locality-sensitive approach, chap. 8, pp. 91–102. SIAM (2000)
47. Pinsker, M.: On the complexity of a concentrator. In: Proceedings of the 7th International Teletrac Conference, pp. 318/1–318/4 (1973)
48. Reingold, O.: Undirected ST-connectivity in log-space. In: Proceedings of the 37th ACM Symposium on Theory of, Computing, pp. 376–385 (2005)
49. Reingold, O., Vadhan, S., Wigderson, A.: Entropy waves, the zig-zag graph product, and new constant-degree expanders and extractors. In: Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer, Science, pp. 3–13 (2000)
50. Seidel, R.: On the all-pairs-shortest-path problem. In: Proceedings of the 24th ACM Symposium on Theory of, Computing, pp. 745–749 (1995)
51. Szegedy, M., Vishwanathan, S.: Locality based graph coloring. In: Proceedings 25th ACM Symposium on Theory of Computing, San Diego, CA, USA, pp. 201–207, May 1993
52. Schneider, J., Wattenhofer, R.: A new technique for distributed symmetry breaking. In: Proceedings of the 29th ACM Symposium on Principles of, Distributed Computing, pp. 257–266 (2010)